ĐẠI HỌC QUỐC GIA

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HCM

BÁO CÁO ĐỒ ÁN 2

ĐỀ TÀI: NHẬN DẠNG CHỮ VIẾT

GVHD: TS. HOÀNG TRANG

SVTH: TRẦN VĂN TRỌNG MSSV: 41204108

Mục lục

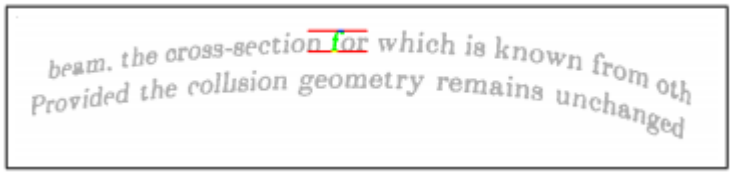
Chương 1 : Kĩ thuật nhận dạng chữ viết hiện nay.

1.1 Sự phát triển của công nghệ nhận dạng chữ viết.

- Ngày nay, nhờ sự phát triển của khoa học kĩ thuật . Hầu hết các lĩnh vực khoa học đều đạt được những thành tựu đáng kể. Trong đó lĩnh vực xử lí hình ảnh và âm thanh đang ngày càng có những tiến bộ đặc biệt là lĩnh vực nhận dạng chữ viết. Không khó khi ta dễ dàng tìm thấy rất nhiều phần mềm nhận dạng chữ trên mạng cho hầu hết các hệ điều hành như window, linux…Những chiếc máy scanner dần được ra đời. Với nhu cầu cuộc sống ngày càng tăng cao kĩ thuật nhận dạng đang dần được áp dụng nhiều trong cuộc sống như : nhận dạng biển số xe, nhận dạng khuôn mặt, nhận dạng vật cảng trên xe tự hành với độ chính xác dần được cải thiện.

1.2 Những vấn đề nan giải.

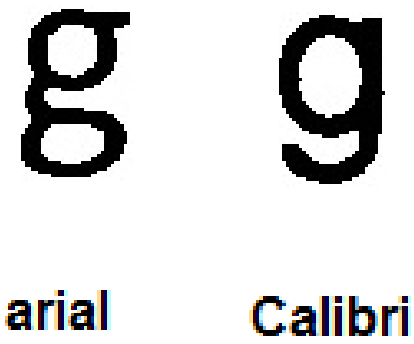
- Riêng trong lĩnh vực nhận dạng chữ viết có rất nhiều chương trình nhận dạng tuy nhiên kết qủa nhận dạng chưa thật sự chính xác, máy scanner có thể quét 1 trang giấy phẳng để có độ chính xác tốt. Nhưng nếu ta muốn nhận dạng một quyển sách ta sẽ xé bìa ra để lấy từng trang và scan ? vì nếu một quyển sách dày sẽ ko thể uốn thẳng được để có độ chính xác kém và các dòng chữ sẽ dễ dàng bị uốn cong. Vấn đề đặt ra làm thế nào để xử lí dòng chữ bị uốn cong ?



Hình 1.1

- Dòng chữ bị uốn cong các đặc trưng sẽ khác so với mẫu đã training.
- Bên cạnh đó ta có thể dùng mẫu chữ nghiên để training tuy nhiên với mỗi gốc nghiên khác nhau sẽ có những đặc trưng vô cùng khác biệt ta sẽ có một lượng mẫu vô cùng lớn.

- Ngày nay nhu cầu giải trí của con người ngày càng tăng cao, số lượng font chữ đang ngày càng được mở rộng với mỗi font chữ khác nhau những chữ giống nhau sẽ có mẫu khác nhau chính vì vậy đòi hỏi người lập trình phải lấy mẫu rất nhiều để training và từ đó sai số tăng lên và thời gian chạy tăng lên là
  không thể tránh khỏi. Vậy với những thuật toán cũ làm thế nào để giảm thiểu thời gian chạy mà vận nhận dạng chính xác được nhiều font chữ ?



Hình 1.2.

Với chữ g đã có một sự khác biệt khá lớn.

- Hầu hết các máy và phần mềm hiện nay đều nhận dạng tốt các văn bản bao gồm toàn chữ nhưng khi gặp các vấn đề như tranh ảnh kí tự lạ thì nó sẽ nhận dạng ra kí tự gì khi trong tập mẫu không có. Vì lượng kí tự là vô cùng lớn. Hãy tưởng tượng trong Microsoft word phần insect symbol đã có rất nhiều kí hiệu đặc biệt vậy để training tất cả chúng ta phải mất thời gian bao lâu.

- Với những mẫu có nhiễu khi ta tách kí tự sẽ có một số chữ dính lấy nhau hoặc các vết lem của nhiễu . Nếu ta lấy giới hạn của vùng liên thông giới hạn dưới rất bé thì ta sẽ tách thêm những đốm nhiễu, để mất chúng ta nên làm những gì.

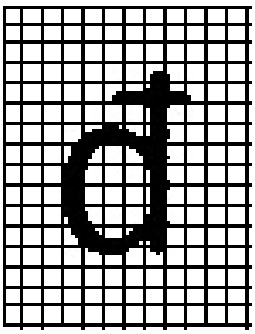
- Tóm lại với rất nhiều vấn đề đặt ra : Với kiến thức bản thân hiện tại rất khó để khắc phục trong một thời gian ngắn. Hi vọng trong tương lai chúng ta có thể khắc phục càng nhiều càng tốt các hạn chế này.

Chương 2 : Các phương pháp trích đặc trưng.

2.1 Phương pháp trọng số vùng.

- Giải thuật : Ta sẽ biến đổi chỉnh kích thước ảnh lại là 24x32 chia ảnh thành các vùng nhỏ hơn ví dụ với ảnh kích thước 23x32 ta sẽ chia thành 192 vùng mỗi vùng có kích thước 2x2 như vậy mỗi đặc trưng sẽ có số điểm đen tối đa là 4 và tối thiểu là 0.

- Giả sử ta có hình sau:



Hình 2.1

- Ta sẽ tính các điểm đen chứa trong mỗi ô số đặc trưng của ảnh sẽ là (24/2)x(32/2) = 192 đặc trưng.

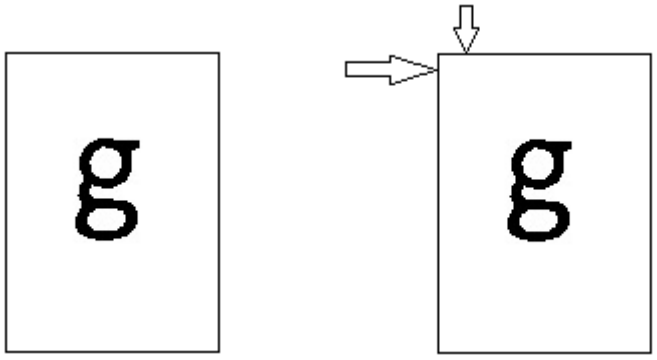
```
for ( int i = 0 ; i < h ; i = i + 2)
        {
          for (int j = 0 ; j < w ; j = j+2)
          {
            for ( int i1 = i; i1 < i+2; i1 ++)
            {
              for ( int j1 = j ; j1 < j+2 ; j1 ++)
              {
```

```
                if ( img.at<uchar>(i1,j1)==0)
                {
                   dem ++;
                }
             }
            }
            F[n]= dem;
            n++;
            dem = 0;
         }
      }
```

- Sử dụng các vòng lặp vào nhau để tính số điểm đen của mỗi vùng vì mỗi vùng có
  kích thước là 2x2 nên mỗi lần lập cho i = i+2 tương tự với j = j +2.

2.2 Phương pháp tham chiếu.

- Giải thuật :



Hình 2.2

- Với phương pháp này ta sẽ tính tổng số điểm đên trên mỗi dòng và mỗi cột, sau đó tính tổng điểm đen trên mỗi đường chéo. Số các đặc trưng là 24+32+24x2+32x2 = 164 đặc trưng .
- Để tính tổng số điểm đen trên các hàng và các cột ta dùng các vòng lặp :

```
for (int i=0; i<h; i++)
```

```
{
   for ( int j = 0; j< w; j++)
      {
         if (img.at<uchar>(i,j) == 0)
               {
                  dem ++;
               }
       }
   F[n] = dem ;
   dem = 0;
   n++;
 }
```

- Riêng với các tính tổng số điểm đen trên hàng chéo khá là phức tạp giả sử ta có ảnh minh hoạ kích thước 8x8 bên dưới mỗi ô ứng với mỗi toạ độ điểm ảnh:

00	10	20	30	40	50	60	70
01	11	…	…				
02	…						
03							
04							
05							
06							
07							…

Hình 2.3

- Ta thấy toạ độ các điểm trong 1 hàng chéo này có tổng không đổi và tăng từ 0 => 7+7 = 14. Ta có đoạn code tính tổng điểm đen như sau :

```
for (int k = 0; k <55 ; k++)
      {
         for ( int i2 = 0 ; i2 < h ; i2++)
```

```
    {
       for ( int j2 = 0 ; j2 <w ; j2 ++)
       {
          if ( (i2 + j2)==k)
          {
             if ( img.at<uchar>(i2,j2) ==0)
             {
                dem ++ ;
             }
          }
       }
    }
  F[n]= dem ;
     dem = 0 ;
     n ++;
  }
```

- Ở đây có số 55 do ảnh ta có kích thước 24x32 nên ta có 23+31 = 54 tổng lớn nhất là 54 bé hơn 55.
- Tương tự với hàng chéo còn lại :

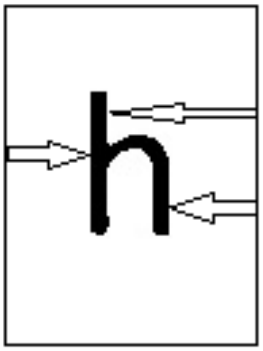
..					05	06	07
						16	17
							27
							..

Hình 2.4

- Toạ độ sẽ tăng dần thêm 1 đơn vị trong chuỗi cho đến khi toạ độ x hoặc y bằng 7 .

2.3 Phương pháp chu tuyến.

- Giải thuật:



Hình 2.5

- Với phương pháp này ta sẽ tính khoảng cách từ biên đến vị trí có điểm đen, lần lượt tính từ trái => phải,phải => trái, trên => dưới,dưới => trên. Như vậy các giá

  trị đặc trưng sẽ nằm trong khoảng từ 0 -> 32 tổng số đặc trưng sẽ là 32+32+24+24

  =112 đặc trưng.

```
for (int a=0; a<h; a++)
        {
          for ( int b =w-1; b>=0; b--)
            {
                  if (img.at<uchar>(a,b) == 0)
                        {
                          F[dem] = w - b-1;
                    dem= dem +1;
                        break ;
                        }
              if ((b ==0)&(img.at<uchar>(a,b) != 0))
```

```
                {
                    F[dem] = 0;
                    dem = dem +1;
                  break ;
                }

            }
        }
```

- Ta dùng các hàm lặp này để tính số khoảng cách lần lượt từ các cạnh đến số điểm đen đầu tiên nếu không có điểm đen ta cho khoảng cách là 0.

2.4 **Phương pháp wavelet haar**.

- Giải thuật:

  Ta có ảnh đầu vào kích thước 24x32 giả sử với phương pháp này ta chia ảnh thành 12 phần nhỏ như hình dưới.Thuật toán wavelet haar áp dụng như sau:

S0	S1	S2	S3
S4	S5	S6	S7
S8	S9	S10	S11

Hình 2.6

- F[0] = Số pixel trong toàn bộ ảnh.
- Chia ảnh thành 12 vùng như trên mỗi vùng là 0,1……,11.
- Gọi Si là số điểm đen của ma trận(trong đó i là số vùng).
- Ma trận lớn kích thước 24x32.
- Tức F0 = S0+S1+S2+S3+…...+S11 (số điểm đen ma trận 24x32).
- Khi đó ta cần tính :

  F1 = S0+S1.

F2=S1+S2.

F3 =S2+ S3.

..

F13= S13.

- Tiếp tục chia nhỏ từng S ra làm 4 phần nhỏ:

S00	S01	S02	…				
			…				

Hình 2.7

- Gọi F11,F12,F13,F14 là diện tích các phần nhỏ trong S1.
- Tương tự ta có F21,F22…….F124.

F14 = S00+S01.

F15=S02+S03.

..

- Tiếp tục chia nhỏ và tính số điểm đen ta được

1+11+12x3+3x4x12 = 192 đặc trưng.

- Để tính số điểm đen từng phần như vậy ta dùng các vòng lập xen lẫn nhau.

```
for (int k=8; k>2 ; k=k/2)
            {
             for (int i=k ; i<=h ; i=i+k)
                 {
                    for (int j=k ; j<=w ; j=j+k)
                    {
                       int n= i-k/2;
                       int m= j-k/2;
```

```
                int t=0;
                s[t]= dem_pixel(img,black_pixel,n-k/2,n,m,m+k/2);
                s[t+1]= dem_pixel(img,black_pixel,n-k/2,n,m-k/2,m);
                s[t+2]= dem_pixel(img,black_pixel,n,n+k/2,m-k/2,m);
                s[t+3]= dem_pixel(img,black_pixel,n,n+k/2,m,m+k/2);
                F[dem]=s[0]+s[1];
                F[dem+1]=s[1]+s[2];
                F[dem+2]=s[3];
                dem = dem+3;
            }
        }
    }
```

- Trong chương trình này ta dùng hàm dem_pixel với ảnh đầu vào là img, biến black_pixel mang giá trị true thì ta sẽ đếm số điểm đen còn mang giá trị false thì ta sẽ đếm số các điểm trắng trong ảnh nhị phân.

-Hàm dem_pixel :

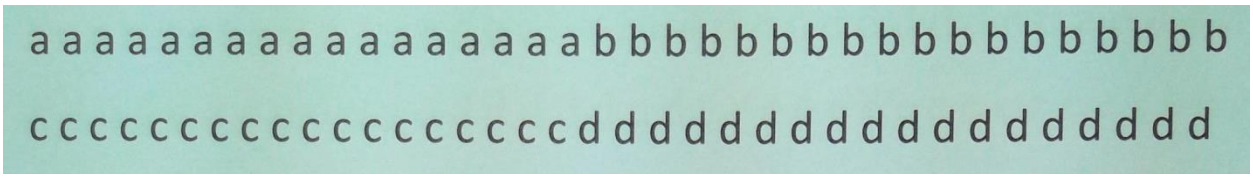
```
int dem_pixel( Mat img, bool  black_pixel, int fromrows, int torows, int fromcols, int tocols)
        {
            int black = 0;
            int write = 0;
            for (int nn=fromrows;nn< torows;++nn)
               {
                  for (int mm=fromcols;mm<tocols; ++mm)
                  {
                     if (img.at<uchar>(nn,mm) == 0)
                      {
                            black ++;
                            else
```

```
                write++;
            }
        }
        if (black_pixel)
        return black;
        else
        return write;
    }
```

Chương 3 : Phương pháp tách chữ đường bao.

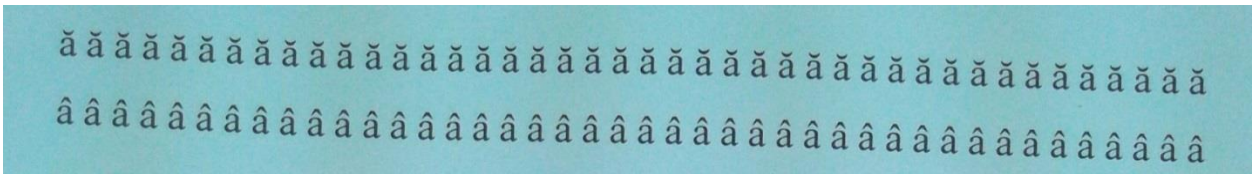
- Chia ảnh theo các vùng liên thông áp dụng với ảnh chứa cả dấu dùng mô hình
  SVM để tách phần dấu và chữ(các khối liên thông).
- **Bước 1**: Xác định các vùng liên thông.
- **Bước 2**: Xắp xếp các vùng liên thông theo thứ tự từ trên xuống.
- **Bước 3** : Nếu ảnh chỉ có 1 vùng liên thông chuẩn hoá ảnh thành 24x32.
- Nếu ảnh có 2 vùng liên thông gọi S(i) là diện tích của vùng liên thông thứ i
- Nếu S(1) > S(2) thì vùng liên thông 2 là dấu (.) và vùng liên thông 1 chuẩn hoá về kích thước 16x16.
- Ngược lại thì phần S2 chuẩn hoá về 24x32 và phần S3 chuẩn hoá về kích thước 8x8.
- Nếu ảnh có 3 vùng liên thông thì S(3) là (.) và chuẩn hoá phần liên thông S2 về 24x32 và phần liên thông S1 chuẩn hoá về kích thước là 8x8.
- Ngôn ngữ có dấu chia làm 3 nhóm :
  +Nhóm 1 gồm : chữ hoa A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z, và chữ thường là a,b,c,d,e,f,g,h,I,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z.
  +Nhóm 2 gồm : ă, â,ê,ô,á,à,ạ,ẹ,ọ,i,ợ,ớ,ò,ó,ờ,ợ….và chữ hoa tương tự.
  +Nhóm 3 gồm : Ặ ,Ậ, Ộ…..và chữ thường tương tự.

+ Thuật toán còn lại chia nhỏ và thực hiện tương tự phương pháp -- -- Chẳng hạn ta có ảnh sau :



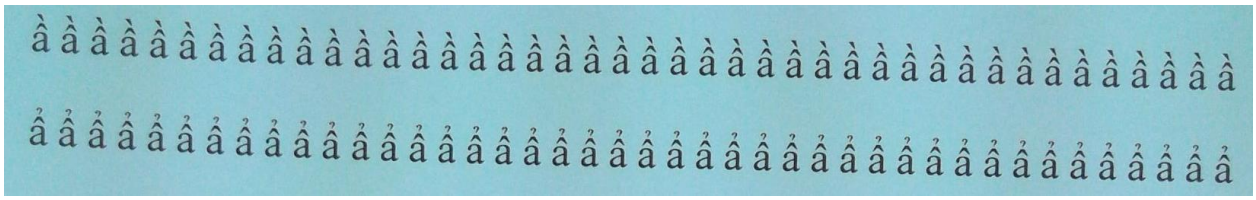
Hình 3.1

- Đây là những chữ chỉ có 1 vùng liên thông. Ta tách chữ bằng cách xác định các điểm đen trong ảnh và phân chia vùng có các điểm đen liên tiếp nhau với vùng không có điểm đen.



Hình 3.2

- Đây là những ảnh có 2 vùng liên thông với những chữ này ta cần nhị phân hoá ảnh sao cho nhiễu trên ảnh là bé nhất vì khoảng cách giữa dấu và chữ rất bé nếu có nhiễu xảy ra trên ảnh thì rất dễ ta sẽ tách cả chữ và dấu vào cùng 1 vùng liên thông với nhau.



Hình 3.3

- Những ảnh có 3 vùng liên thông.
- Ta dùng hàm

```
findContours(img_binary, contours, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
```

để tìm các vùng liên thông và hàm :

```
boundRect[i] = boundingRect( Mat (contours[i]));
```

để tách các vùng liên thông nhau ra.

Chương 4 : Kiến thức về SVM.

4.1 Định nghĩa svm và lịch sử hình thành.

- Thuật toán SVM ban đầu được tìm ra bởi Vladimir N. Vapnik và dạng chuẩn hiện nay sử dụng lề mềm được tìm ra bởi Vapnik và Corinna Cortes năm 1995.
- SVM (Support Vector Machine) là một khái niệm trong thống kê và khoa học máy tính cho một tập hợp các phương pháp học có giám sát liên quan đến nhau để phân loại và phân tích hồi quy. SVM là một thuật toán phân loại nhị phân, SVM nhận dữ liệu vào và phân loại chúng vào hai lớp khác nhau. Với một bộ các ví dụ luyện tập thuộc hai thể loại cho trước, thuật toán luyện tập SVM xây dựng một mô hình SVM để phân loại các ví dụ khác vào hai thể loại đó.SVM là mô hình xây dựng một siêu phẳng hoặc một tập hợp các siêu phẳng trong một không gian nhiều chiều hoặc vô hạn chiều, có thể được sử dụng cho phân loại, hồi quy, hoặc các nhiệm vụ khác. Để phân loại tốt nhất thì phải xác định siêu phẳng (Optimal hyperplane) nằm ở càng xa
  các điểm dữ liệu của tất cả các lớp (Hàm lề) càng tốt, vì nói chung lề càng lớn thì sai số tổng quát hóa của thuật toán phân loại càng bé. Trong nhiều trường hợp, không thể phân chia các lớp dữ liệu một cách tuyến tính trong một không gian ban đầu được dùng để mô tả một vấn đề. Vì vậy, nhiều khi cần phải ánh xạ các điểm dữ liệu trong không gian ban đầu vào một không gian mới nhiều chiều hơn, để việc phân tách chúng trở nên dễ dàng hơn trong không gian mới.

4.2 SVM tuyến tính.

- Ta có một tập huấn luyện D gồm n điểm có dạng:
  $D= \{(xi,yi)|xi \epsilon Rp, yi \epsilon\{-1,1\}^{n}_{i=1}$. với y_i mang giá trị 1 hoặc -1, xác định lớp của điểm xi mỗi xi là một vectơ thực p-chiều. Ta cần tìm siêu phẳng có lề

lớn nhất chia tách các điểm có yi = 1 và các điểm có yi= -1. Mỗi siêu phẳng đều có thể được viết dưới dạng một tập hợp các điểm x thỏa mãn: w.x-b = 0 với kí hiệu (.) cho tích vô hướng và w là một vectơ pháp tuyến của siêu phẳng. tham số $\frac{b}{||w||}$ xác định khoảng cách giữa gốc tọa độ và siêu phẳng theo hướng vectơ pháp tuyến w.

- Chúng ta cần chọn w và b để cực đại hóa lề, hay khoảng cách giữa hai siêu mặt song song ở xa nhau nhất có thể trong khi vẫn phân chia được dữ liệu. Các siêu mặt ấy được xác định bằng : w.x- b = 1 và w.x – b = -1.
- Để ý rằng nếu dữ liệu huấn luyện có thể được chia tách một cách tuyến tính, thì ta có thể chọn hai siêu phẳng của lề sao cho không có điểm nào ở giữa chúng và sau đó tăng khoảng cách giữa chúng đến tối đa có thể. Bằng hình học, ta tìm được khoảng cách giữa hai siêu phẳng là $\frac{2}{||w||}$. Vì vậy ta muốn cực tiểu hóa giá trị ||w||. Để đảm bảo không có điểm dữ liệu nào trong lề, ta thêm vào các điều kiện sau, với mỗi i ta có :

  w.xi –b >1 cho xi thuộc lớp thứ nhất và w.xi-b < -1 cho xi thuộc lớp thứ 2.

- Ta có thể viết gọn lại như sau với mọi 1< i < n :

  $y.(w.xi – b) \geq 1$.

- Tóm lại ta có điều kiện tối ưu sau:

  cực tiểu ||w|| với điệu kiện với mọi 1<i<n ta có $y.(w.xi-b) \geq 1$.

4.2.1 **SVM dạng ban đầu**.

- Bài toán tối ưu ở mục trên tương đối khó giải vì hàm mục tiêu phụ thuộc vào ||w||, là

  một hàm có khai căn. Tuy nhiên có thể thay ||w|| bằng hàm mục tiêu $\frac{1}{2}||w||^2$ (hệ số 1/2 để tiện cho các biến đổi toán học sau này) mà không làm thay đổi lời giải (lời giải của bài toán mới và bài toán ban đầu có cùng w và b). Đây là một bài toán quy hoạch toàn phương. Cụ thể hơn:
- Cực tiểu hóa (theo w,b)

$\frac{1}{2}||w||^2$

- Với điều kiện : $y.(w.xi - b) \geq 1$.
- Bằng cách thêm các nhân tử α Lagrange , bài toán trên trở thành:

$$\min_{w,b} \max_{\alpha \geq 0} \{\frac{1}{2}||w||^2 - \sum_{n} \alpha i[yi.(w.xi - b) - 1]\}$$

nghĩa là ta cần tìm một điểm yên ngựa. Khi đó, tất cả các điểm không nằm trên lề, nghĩa là yi.(w.xi –b) – 1 >0 đều không ảnh hưởng đến giá trị hàm mục tiêu vì ta có thể chọn $\alpha i = 0$.

- Có thể giải bài toán này bằng các kĩ thuật thông thường cho quy hoạch toàn phương. Theo điều kiện Karush–Kuhn–Tucker, lời giải có thể được viết dưới dạng tổ hợp tuyến tính của các vectơ luyện tập :

$$w = \sum_{n} \alpha i.xi.yi$$

- Chỉ có một vài αi nhận giá trị lớn hơn 0. Các điểm xi tương ứng là các vectơ hỗ trợ nằm trên lề và thỏa mãn yi.(w.xi – b) =1. Từ điều kiện này, ta nhận thấy :

w.xi -1 = 1/yi = yi ⇔ b = w.xi –yi từ đó ta suy ra được giá trị b. Trên thực tế, một cách thức tốt hơn để tính b là tính giá trị trung bình từ tất cả N_{SV} vectơ hỗ trợ:

$$b = \frac{1}{Nsv} \sum_{n=1}^{Nsv} (w.xi - yi)$$

4.2.2 Dạng đối ngẫu.

- Nếu viết điều kiện phân loại dưới dạng đối ngẫu không điều kiện thì sẽ dễ dàng nhận thấy siêu phẳng với lề lớn nhất, và do đó nhiệm vụ phân loại, chỉ phụ thuộc vào các điểm luyện tập nằm trên lề, còn gọi là các vectơ hỗ trợ.
- Vì $||w||^2 = w.w$ và $w = \sum_{i=1}^{n} \alpha i.yi.xi$, ta nhận thấy bài toán đối ngẫu của SVM là

chính là bài toán tối ưu hóa sau (cực đại hoá theo αi):

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha i - \frac{1}{2}.\sum_{i,j} \alpha i.\alpha j.xi^T.xj.yi.yj = \sum_{i=1}^{n} \alpha i - \sum_{i,j} \frac{1}{2}.\alpha i.\alpha j.yi.yj.k(xi,xj)$$ với điều

kiện (với mọi I = 1,2,……,n) $\alpha i \geq 0$ và điều kiện sau ứng với việc cực tiểu hóa theo b.

$$\sum^{n} \alpha i. yi = 0$$

- Ở đây hàm hạt nhân được định nghĩa là k(xi,xj) = xi.xj.
- Sau khi giải xong, có thể tính w từ các giá trị α tìm được như sau: w = $\sum_i yi. xi. \alpha i$

4.3 SVM với lề mềm.

4.3.1 SVM dạng ban đầu.

- Năm 1995, Corinna Cortes và Vladimir N. Vapnik đề xuất một ý tưởng mới cho phép thuật toán gán nhãn sai cho một số ví dụ luyện tập.[2] Nếu không tồn tại siêu phẳng nào phân tách được hai lớp dữ liệu, thì thuật toán lề mềm sẽ chọn một siêu phẳng

  phân tách các ví dụ luyện tập tốt nhất có thể, và đồng thời cực đại hóa khoảng cách giữa siêu phẳng với các ví dụ được gán đúng nhãn. Phương pháp này sử dụng các biến bù εi , dùng để đo độ sai lệch của ví dụ xi.

  $Yi.(w.xi - b) \geq 1 - \varepsilon i , 1 \leq i \leq n$

- Hàm mục tiêu có thêm một số hạng mới để phạt thuật toán khi khác không, và bài toán tối ưu hóa trở thành việc trao đổi giữa lề lớn và mức phạt nhỏ. Nếu hàm phạt là tuyến tính thì bài toán trở thành: $\min_{w,\varepsilon,b} . \frac{1}{2} . ||w||^2 + c. \sum_{i=1}^{n} \varepsilon i$ với điệu kiện với mọi I =(1,2,……,n)

  $Yi.(w.xi - b) \geq 1 - \varepsilon i , \varepsilon i \geq 0$

- Có thể giải bài toán trên bằng nhân tử Lagrange tương tự như trường hợp cơ bản ở trên. Bài toán cần giải trở thành:

  $\min_{w,\varepsilon,b} \max_{\alpha,\beta} (\quad ½ . ||w||^2 + C \sum_{i=1}^{n} \varepsilon i \quad - \quad \sum_{i=1}^{n} \alpha i [yi. (w. xi - b) - 1 + \varepsilon i] - \sum_{i=1}^{n} \beta i. \varepsilon i \}$ với $\alpha i, \beta i > 0$.

4.3.2 Dạng đối ngẫu

- Cực đại hoá theo αi:

$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha i - \frac{1}{2}. \sum_{i,j} \alpha i. \alpha j. k(xi. xj). yi. yj$ với điều kiện (với mọi I = 1,2,…..,n)

$0 \leq \alpha i \leq C.$

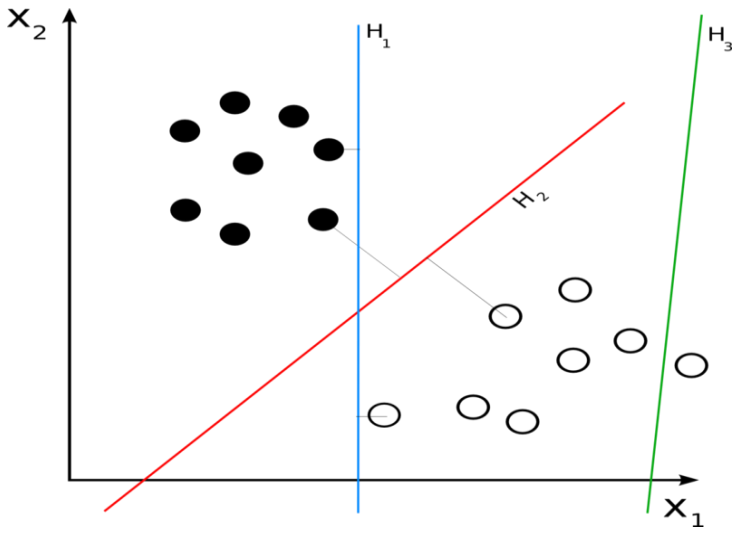
và

$$\sum^{n} \alpha i. yi = 0$$

- Ưu điểm của việc dùng hàm phạt tuyến tính là các biến bù biến mất khỏi bài toán đối ngẫu, và hằng số C chỉ xuất hiện dưới dạng một chặn trên cho các nhân tử Lagrange.
- Cách đặt vấn đề trên đã mang lại nhiều thành quả trong thực tiễn, và Cortes và Vapnik đã nhận được giải Paris Kanellakis của ACM năm 2008 cho đóng góp này. Các hàm phạt phi tuyến cũng được sử dụng, đặc biệt là để giảm ảnh hưởng của các trường hợp ngoại lệ, tuy nhiên nếu không lựa chọn hàm phạt cẩn thận thì bài toán trở thành không lồi, và việc tìm lời giải tối ưu toàn cục thường là rất khó.

4.4 Ứng dụng SVM trong thư viện openCV.

- Tôi có một khôn gian có nhiều điểm và các kí hiệu như sau:



Hình 4.1

- Phân tích

- y_i: Đây là các lớp (Bản lề) chứa các điểm dữ liệu x_i. Ở ví dụ này mang giá trị 1 và -1.
- x_i: Là một vector thực nhiều chiều (p chiều).
- Nhiệm vụ là cần phải tìm một siêu phẳng (Optimal hyperplane) có lề lớn nhất chia tách các điểm dữ liệu có ban đầu để huấn luyện và các điểm sau này. Mỗi siêu phẳng (Optimal hyperplane) đều có thể được viết dưới dạng một tập các điểm thỏa mãn:
- w.x – b = 0
- w: Là một vectơ pháp tuyến của siêu phẳng (Optimal hyperplane).
- Tham số $^{b}/_{\|w\|}$ xác định khoảng cách giữa gốc tọa độ và siêu phẳng theo hướng vectơ pháp tuyến w.Như bạn có thể thấy ở hình s1. Tôi giả sử có tới 3 siêu phẳng (Optimal hyperplane) là H1 (Xanh dương), H2(Đỏ), H3 (Xanh lá). H3 sẽ bị loại đầu tiên vì không thể phân loại các điểm huấn luyện cho trước. H1 bị loại vì khoảng cách từ các điểm Support Vector đến siêu phẳng (Optimal hyperplane) chưa phải là cực đại. H2 là siêu phẳng cần tìm. Lúc này các siêu phẳng đó được xác định: w.x –b =1

và w.x – b = -1

- Các điểm dữ liệu cho trước nằm trên các siêu phẳng song song được gọi là Support - vector.

- Thiết lập dữ liệu huấn luyện

```
1. float labels[4] = {1.0, -1.0, -1.0, -1.0};
2. float trainingData[4][2] = {{501, 10}, {255, 10}, {501, 255}, {10, 501}};
```

- Dữ liệu được huấn luyện được tạo thành bởi một tập điểm có giá trị 2D. Ở đoạn code trên, ta có: 2 lớp: Lớp thứ nhất mang giá trị -1, và lớp thứ hai mang giá trị 1 (lables).

- 4 điểm dữ liệu (Điểm được dùng để huấn luyện mô hình SVM) cho trước ứng với giá trị của lớp: (501, 10) thuộc lớp thứ hai, và 3 điểm còn lại thuộc lớp thứ nhất.

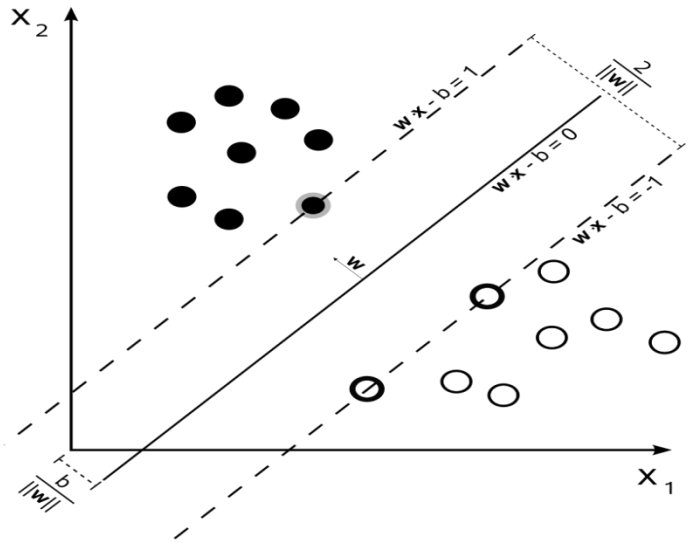
```
1. Mat trainingDataMat(4, 2, CV_32FC1, trainingData);
2. Mat labelsMat(4, 1, CV_32FC1, labels);
```

- Thiết lập thông số của SVM

```
1. // Set up SVM's parameters
2. Ptr<ml::SVM> svm = ml::SVM::create();
3. svm->setType(ml::SVM::C_SVC);
4. svm->setKernel(ml::SVM::LINEAR);
5. svm->setTermCriteria(cv::TermCriteria(CV_TERMCRIT_ITER, 100, 1e-6));
```

- Phân tích

- Type SVM: Chọn ml::SVM::C_SVC để có thể được sử dụng để phân



Hình 4.2

loại nhiều (n) lớp (n>2). Tham số này được xác định trong thuộc tính ml::SVM::Params.svmType. Lưu ý: Chức năng quan trọng của loại SVM CvSVM::C_SVC là tách hoàn hảo của các lớp học (Tức là khi dữ liệu đào tạo là không bị phân chia qua các lớp khác).

- Type SVM kernel: Là một tham số này được xác định trong thuộc tính ml::SVM.KernelType.
- Termination criteria of the algorithm: Là một tham số này được định nghĩa trong một cấu trúccv::TermCriteria.

- Huấn luyện SVM

Để xây dựng mô hình huấn luyện SVM, ta sử dụng phương thức CvSVM::train().

```
1. CvSVM SVM;
2. SVM.train(trainingDataMat, labelsMat, Mat(), Mat(), params);
```

Như vậy ta đã huấn luyện xong cho mô hình SVM của ví dụ ở trên.

- Phân loại lớp (Bản lề)

Phương pháp CvSVM::predict() được sử dụng để phân loại một mẫu đầu vào bằng cách sử dụng một SVM được huấn luyện.

```
1. Vec3b green(0,255,0), blue (255,0,0);
2.
3. for (int i = 0; i < image.rows; ++i) {
4.     for (int j = 0; j < image.cols; ++j)
5.     {
6.     Mat sampleMat = (Mat_<float>(1,2) << i,j);
7.     float response = SVM.predict(sampleMat);
8.
```

```
9.      if (response == 1)
10.         image.at<Vec3b>(j, i)  = green;
11.     else if (response == -1)
12.         image.at<Vec3b>(j, i)  = blue;
13.     }
14.  }
```

Chương 5: Phương pháp xử lí bố cục văn bản đơn giản.

- Ở đây ta dùng phương pháp biểu đồ chiếu phương pháp này chỉ áp dụng cho các văn

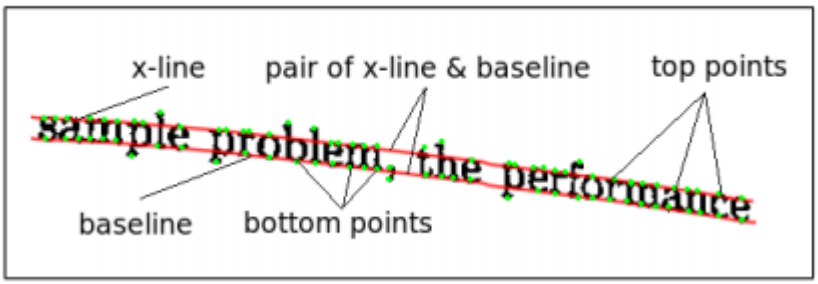
  bản gồm nhiều cột với văn bản có ảnh thì phương pháp này thiếu đi độ chính xác của chúng.

- Ta thấy với văn bản bên dưới khi ta chiếu các điểm đen theo cột thì ngay tại điểm phân tách 2 cột văn bản sẽ bằng không. Ta sẽ cắt tại điểm này để tạo thành 2 văn bản và tiến hành nhận dạng tương tự khi chiếu sang chiều ngang. Khi đó nếu phân cắt ra ta sẽ được 4 văn bản riêng biệt nhau tránh bị lẫn hàng và cột.

Hình 5.1

Chương 6 : Các phương pháp xử lí ảnh bị uốn cong.

- Phương pháp coupled snakelet.

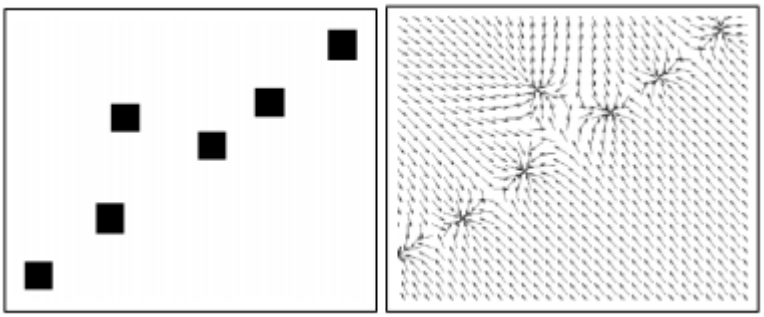


Hình 6.1

- Đây là phương pháp được đánh giá có độ chính xác nhất hiện nay.
- Thuật toán : Ta sẽ có một cặp đường gọi là x-line, đường này sẽ nối các đỉnh của các chữ lại với nhau tạo thành một dải băng rôn bị uốn cong theo hàng

chữ, để xác địnhđược cặp x-line này ta cần xác định các điểm đỉnh (top points) hoặc dưới cùng

(bottom points) của các chữ từ các vector lân cận ta nối các điểm đó lại với nhau bằng các đường (baseline) tạo thành cặp đường x-line.

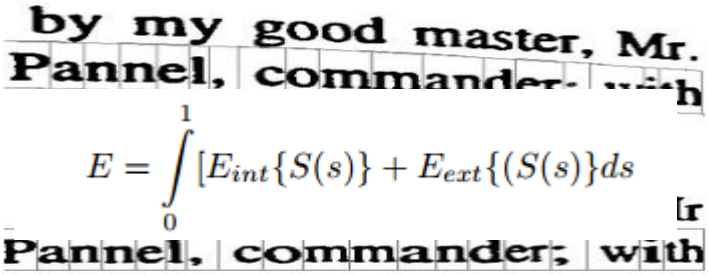


Hình 6.2

- Sử dụng các dòng vector gradient để nối các top points với nhau và các bottom points với nhau.

- Hàm tính gradient.

- Phương pháp này tỉ lệ chính xác có thể đạt 90.76 %.
- Phương pháp Robust Estimation.



Hình 6.3

- Hàng chữ bị uốn cong đã được khôi phục lại.

Chương 7 : Công cụ và một số hàm phụ dùng trong nhận dạng chữ viết.

- Ta dùng chương trình qt creator kết hợp với thư viện openCV trong hệ điều hành Ubuntu để thực hiện đề tài, viết bằng ngôn ngữ C++.
- Thư viện openCV là thư viện dùng trong xử lí ảnh và video.
- Một số hàm phụ dùng để nhận dạng chữ viết:

  + Hàm liệt kê file có trong thư mục :

  Ta có thể dùng symtem(lệnh trong Ubuntu);

  Tuy nhiên, để có thể thuận tiện hơn ta dùng hàm liệt kê file sau :

```
int lietkefile(string dir, vector<string> &files)
                        {
                            DIR *dp;
                            struct dirent *dirp;
                            if((dp = opendir(dir.c_str())) ==
NULL)
                                {
                                   return errno;
                                }
                             while ((dirp = readdir(dp)) !=
NULL)
                                {
                             files.push_back(string(dirp-
>d_name));
                                }
                             closedir(dp);
                             return 0;
                            }
```

- Hàm viết chuỗi vào file txt:

```
File.open("/home/trong/Desktop/vanban.txt",ios::out|ios::in
);
```

```
for (int o=0 ; o<dem ; o++)
            {
                File << F[o];
                File <<" ";
            }
              File << endl ;

File.close();
```

Chương 8 : Tài liệu tham khảo

Coupled Snakelet Model for Curled Textline Segmentation of Camera-Captured Document Images.

PERFORMANCE EVALUATION OF CURLED TEXTLINE SEGMENTATION ALGORITHMS ON CBDAR 2007 DEWARPING CONTEST DATASET (Syed Saqib Bukhari 1, Faisal Shafait 2 and Thomas M. Breuel 1).

Segmentation of Curled Textlines using Active Contours Syed Saqib Bukhari 1, Faisal Shafait 2, Thomas M. Breuel 1 Image Understanding and Pattern Recognition (IUPR) Research Group 1 Department of Computer Science, Technical University of Kaiserslautern, Germany 2 German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany fbukhari, tmbg@informatik.uni-kl.de,ffaisal.shafaitg@dfki.de,

Phụ lục

```
#include <iostream>
#include "extract_char.h"
#include <dirent.h>

/////***** include opencv lib *****/////
```

```
#include <opencv2/highgui.hpp>
#include <opencv/ml.h>
#include <opencv/cv.h>
#include <opencv/cxcore.h>
#include <opencv2/imgproc.hpp>
#include <fstream>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <vector>
#include <string>
#include "string.h"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv ;
int lietkefile(string dir, vector<string> &files);
int extract_char (std::string input_file) ;
int main()
{
    string input ;
    const char *filename = NULL;
             input = "/home/trong/Desktop/anhdactrung/";
             filename = input.c_str();
             string dir = string(filename);
             vector<string> files = vector<string>();
             lietkefile(dir,files);
            for (unsigned int z = 0;z < files.size();z++)
                {
```

```
                if ( (files[z] != ".")&(files[z] != ".."))
                 {

                    files[z] = files[z];
                    extract_char(files[z]);
                }
            }
}
  int lietkefile(string dir, vector<string> &files)   {
       DIR *dp;
       struct dirent *dirp;
       if((dp = opendir(dir.c_str())) == NULL)









 {
            // cout << "file rong(" << errno << ") dang mo "
<< dir << endl;
             return errno;
          }
          while ((dirp = readdir(dp)) != NULL)
               {
                files.push_back(string(dirp->d_name));
                }
          closedir(dp);
           return 0;
           }
```

```
int extract_char (std::string input_file) {

    /////***** Load Input Image *****/////

    const char* filename = NULL;
    string
str_temp="/home/trong/Desktop/anhdactrung/"+input_file+".jpg";
    string  str_file_name=str_temp.substr(0,str_temp.length()-
4);
    cout << "File can chuyen: " << str_file_name << endl ;
    filename = str_file_name.c_str();
    Mat img_load = cv::imread(filename);
    if(!img_load.data){
        cout << "Error.No image input !" << endl;
    }
    /////***** RGB Image To Gray Image *****/////
    Mat img_gray;
    cv::cvtColor(img_load,img_gray,CV_RGB2GRAY);
    /////***** Gray Image To Binary Image *****/////
    Mat img_binary;
    cv::      adaptiveThreshold(img_gray,img_binary,      255,
CV_ADAPTIVE_THRESH_MEAN_C,CV_THRESH_BINARY, 35, 5);
    //cv::threshold(img_gray,     img_binary,     0,     255,
CV_THRESH_OTSU+CV_THRESH_BINARY);
    cv::imwrite("img_binary.jpg",img_binary);

    vector<vector<Point> > contours;
    cv::findContours(img_binary,    contours,    CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
```

```
    vector<Rect> boundRect( contours.size() );
    for( int i = 0; i < contours.size(); i++ )
    {
        boundRect[i] = boundingRect( Mat (contours[i]));

        /// dung de tach ky tu cho huan luyen
        if((boundRect[i].width>5)&&(boundRect[i].height<20)) {
            Mat img_word_in = cv::imread("img_binary.jpg");
            Mat                    img_char                    =
cv::Mat(img_word_in,boundRect[i]).clone();

            stringstream strs;
            strs << i;
            string                                    temp_str
="/home/trong/Desktop/anhkitu/"+input_file+"_char_"+
strs.str()









+".jpg";
            char* char_name =(char*)temp_str.c_str();
            imwrite(char_name,img_char);
        }
    }
    cout<<"finished file: "<<str_file_name<<endl;
    return 0;
```

}

Training và SVM.

```
#include <iostream>
#include "extract_char.h"
#include <dirent.h>

/////***** include opencv lib *****/////

#include <opencv2/highgui.hpp>
#include <opencv/ml.h>
#include <opencv/cv.h>
#include <opencv/cxcore.h>
#include <opencv2/imgproc.hpp>
#include <fstream>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <vector>
#include <string>
#include "string.h"
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv ;
int lietkefile(string dir, vector<string> &files);
int extract_char (std::string input_file) ;
int main()
```

```
{
    string input ;
    const char *filename = NULL;
             input = "/home/trong/Desktop/anhdactrung/";
             filename = input.c_str();
             string dir = string(filename);
             vector<string> files = vector<string>();
             lietkefile(dir,files);
            for (unsigned int z = 0;z < files.size();z++)

{
                if ( (files[z] != ".")&(files[z] != ".."))
                 {

                    files[z] = files[z];
                    extract_char(files[z]);
                }
            }
}
  int lietkefile(string dir, vector<string> &files)   {
       DIR *dp;
       struct dirent *dirp;
       if((dp = opendir(dir.c_str())) == NULL)
          {
            // cout << "file rong(" << errno << ") dang mo "
<< dir << endl;
             return errno;
          }
          while ((dirp = readdir(dp)) != NULL)
               {
```

```
            files.push_back(string(dirp->d_name));
            }
        closedir(dp);
         return 0;
         }

int extract_char (std::string input_file) {

    /////***** Load Input Image *****/////

    const char* filename = NULL;
    string
str_temp="/home/trong/Desktop/anhdactrung/"+input_file+".jpg";
    string  str_file_name=str_temp.substr(0,str_temp.length()-
4);
    cout << "File can chuyen: " << str_file_name << endl ;
    filename = str_file_name.c_str();
    Mat img_load = cv::imread(filename);
    if(!img_load.data){
        cout << "Error.No image input !" << endl;
    }
    /////***** RGB Image To Gray Image *****/////
    Mat img_gray;
    cv::cvtColor(img_load,img_gray,CV_RGB2GRAY);
    /////***** Gray Image To Binary Image *****/////
    Mat img_binary;
    cv::      adaptiveThreshold(img_gray,img_binary,      255,
CV_ADAPTIVE_THRESH_MEAN_C,CV_THRESH_BINARY, 35, 5);
    //cv::threshold(img_gray,     img_binary,     0,     255,
CV_THRESH_OTSU+CV_THRESH_BINARY);
```

```
    cv::imwrite("img_binary.jpg",img_binary);

    vector<vector<Point> > contours;
    cv::findContours(img_binary,    contours,    CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

vector<Rect> boundRect( contours.size() );
    for( int i = 0; i < contours.size(); i++ )
    {
        boundRect[i] = boundingRect( Mat (contours[i]));

        /// dung de tach ky tu cho huan luyen
        if((boundRect[i].width>5)&&(boundRect[i].height<20)) {

  Mat img_word_in = cv::imread("img_binary.jpg");
            Mat                    img_char                    =
cv::Mat(img_word_in,boundRect[i]).clone();

            stringstream strs;
            strs << i;
            string                                      temp_str
="/home/trong/Desktop/anhkitu/"+input_file+"_char_"+
strs.str() +".jpg";
            char* char_name =(char*)temp_str.c_str();
            imwrite(char_name,img_char);
        }
    }
    cout<<"finished file: "<<str_file_name<<endl;
    return 0;
```

```
}
```